1986-11-15

Information resource management in the DCSPLANS branch of the U.S. Army Military Personel Center

Dolk, Daniel R.

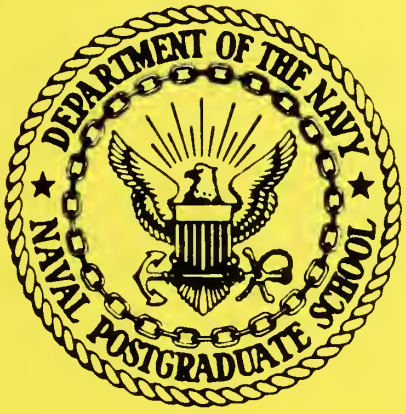
Monterey, California. Naval Postgraduate School

LIBRARY
RESEARCH REPORTS DIVISION
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NPS-54-86-016

NAVAL POSTGRADUATE SCHOOL

Monterey, California



INFORMATION RESOURCE MANAGEMENT IN THE DCSPLANS BRANCH OF THE U.S. ARMY MILITARY PERSONNEL CENTER

by

Daniel R. Dolk

November 1986

Approved for public release; distribution unlimited.

Prepared for: U.S. Army Military Personnel Center
Alexandria, VA 22332

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM. R. C. Austin
Superintendent

David A. Schrady
Provost

The research summarized herein was sponsored by the U. S. Army Military Personnel Center.

This report was prepared by:

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S) NPS54-86-016	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 54	7a NAME OF MONITORING ORGANIZATION
ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b ADDRESS (City, State, and ZIP Code)
NAME OF FUNDING / SPONSORING ORGANIZATION U. S. Military Personnel Center 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MIPR #I 32-12-85
ADDRESS (City, State, and ZIP Code) 200 Stovall St. Attn: DAPC-PLF Alexandria, VA 22332	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

TITLE (Include Security Classification)
Information Resource Management in the DCSPLANS Branch of the U. S. Army Military Personnel Center (Unclassified)

PERSONAL AUTHOR(S)
Daniel R. Dolk

TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
	FROM 10/84 TO 09/85	1986 November 15	

SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Information resource management, data administrator, information resource dictionary system, relational database, structured modeling

ABSTRACT (Continue on reverse if necessary and identify by block number)
DCSPLANS currently suffers from data integrity problems which compromise the validity of the manpower models which they maintain. These problems arise from having to use data maintained by various external agencies as well as a lack of corporate history due to frequent military personnel turnover. This report examines the alternate of information resource management (IRM) as a solution to these problems. IRM is considered from the organizational and technological perspectives. Organizationally, the role of DCSPLANS within U. S. Army MilPerCen is identified and implementation of a civilian billet for data administration recommended as a first step towards controlling data resources. Technologically, a relational dictionary system is designed to support this data administrator. The dictionary is compatible with emerging Federal standards. Examples of how the dictionary can be used within DCSPLANS are presented. Finally, structured modeling is introduced as an integrative modeling discipline which coordinates data and modeling resources via the IRM function.

DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
NAME OF RESPONSIBLE INDIVIDUAL Daniel R. Dolk	22b TELEPHONE (Include Area Code) 408-646-2260 22c OFFICE SYMBOL

FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete
SECURITY CLASSIFICATION OF THIS PAGE

INFORMATION RESOURCE MANAGEMENT IN THE DCSPLANS BRANCH OF THE U. S. ARMY MILITARY PERSONNEL CENTER

by

Daniel R. Dolk

November 1986

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The Deputy Chief of Staff for Plans, U.S. Army Military Personnel Center (DCSPLANS MILPERCEN), has expressed a need for increased control and administration of its information resources. This problem is particularly manifested by DCSPLANS' distrust of the data which drives the many manpower models for which they are responsible. A great deal of time and effort have gone into the development of these sophisticated models which have a significant impact on U.S. Army manpower forecasts and policies. Although the intricacies and behavior of the models themselves are well understood, the validity of the results are very sensitive to the integrity of the underlying data. Data that are not current, for example, can lead to manpower projections which are misleading and eventually result in suboptimal decisions adversely affecting the U.S. Army.

The purpose of this project has been to develop an approach to information resource management (IRM) in DCSPLANS which will result in more control over these resources and subsequently facilitate more effective use of manpower models. Two dimensions are considered with respect to implementing IRM: organizational and technological. The organizational aspect is involved with identifying a permanent position within DCSPLANS dedicated to IRM functions. This concept is discussed in Section 2. The technological aspect involves the design and development of an information resource dictionary system (IRDS) as a software tool which supports IRM. The details of this system are described in Section 3. A summary of our recommendations is provided in Section 4.

The work reported herein involved four thesis projects by Masters students at the Naval Postgraduate School: Major Richard E. Broome [Broome 1985], Major Robert M. DiBona [DiBona 1985], Major Robert A. Kirsch II [Kirsch 1985], and Major Alan F. Noel, Jr. [Noel 1985]. Major Broome conducted a broad information requirements analysis of MILPERCEN, in general, and of DCSPLANS in particular. His conclusion that a permanent IRM billet be assigned to DCSPLANS as a preliminary step in implementing IRM is discussed in the next section. Major Noel and Major Kirsch both worked on development of a dictionary system to support this IRM function. Major Noel built a prototype demonstrating the feasibility of this system and Major Kirsch expanded this prototype to be compatible with the emerging Federal standards for dictionary systems. Major DiBona analyzed data validation requirements for DCSPLANS and showed how these could be incorporated into an active dictionary system. The results of these three efforts are synthesized in Section 3.

2. ORGANIZATIONAL ASPECTS

The heart of the information resource problems facing DCSPLANS lies in certain organizational inefficiencies which exist in MILPERCEN. The sources of these inefficiencies have been detailed very effectively in Major Broome's thesis [Broome 1985] and are summarized below. After a synopsis of the problems, short-term and long-term approaches to solving these problems are discussed.

2.1 Existing Problems in DCSPLANS

Figure 2-1 shows the overall information model for DCSPLANS. Two major areas of information resource problems have been identified in the DCSPLANS environment: data integrity and information resource control. Each of these will be examined in more detail.

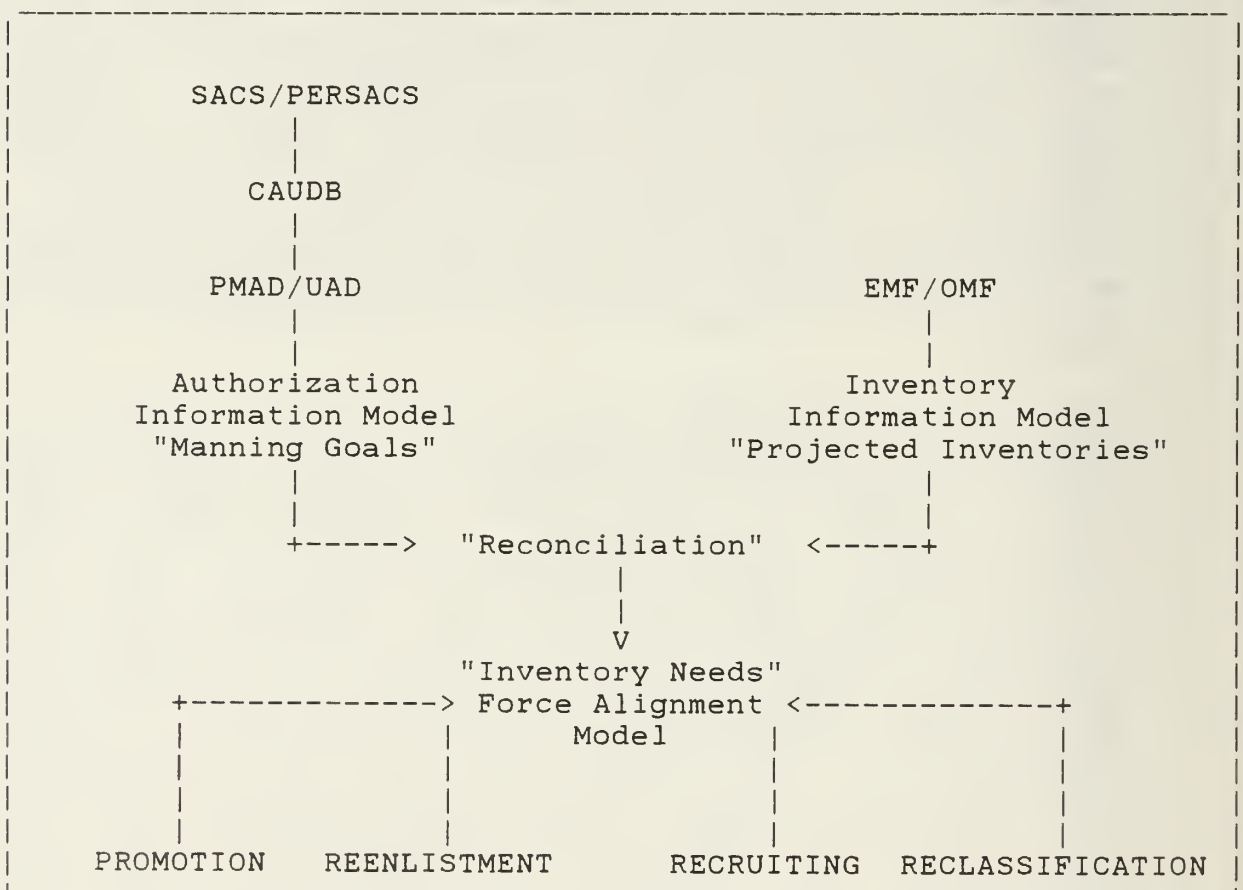


Figure 2-1: The Overall Information Model

2.1.1 Data Integrity of the Overall Information Model

One of the major symptoms of trouble in the DCSPLANS environment is the almost universal acknowledgement by personnel that they "don't trust the data" which they must use to drive their manpower models. Part of the problem is related to the synchronization of updates to the various input files. Since this data is derived from diverse sources, many of which are beyond DCSPLANS' control, there is a significant effort required to validate the currency of the data.

For example, consider the authorization information model shown in Figure 2-1. The Correctable Authorizations DataBase (CAUDB) provides a single source of approved authorizations for managing the force. It is a MILPERCEN product which is a "scrubbed" version of the Personnel Structure and Composition System (PERSACS) which serves as the primary source for authorization data at the Department of the Army level. Interviews with DCSPLANS branch chiefs reveal a common distrust of the PERSACS data and a subsequent reliance on the MILPERCEN-generated PMAD/UAD extracts from the CAUDB instead. The source of this distrust is the belief that the systems which feed the creation of SACS/PERSACS are flawed, untimely, and inaccurate.

The solution adopted by MILPERCEN in the form of the CAUDB is a common one in cases where data integrity is questionable, that is, create redundant data which is under local control. Although this may more effectively serve the needs of the local user community, it causes broader organizational inefficiencies in that redundant effort is required to maintain the duplicated data and data inconsistencies between the two databases inevitably arise. This may, in turn, lead to intra-organizational rivalry concerning ownership and maintenance of the data in question.

The proliferation of redundant data is an all too common characteristic of file-oriented information processing environments. Eventually, the costs of supporting this phenomenon become too high and organizations must invest in some form of data management which facilitates the sharing and maintenance of data resources in an integrated environment.

2.1.2 Lack of Continuous "Corporate History"

Another serious information resource problem in DCSPLANS is the lack of a continuous "corprate history". Because of the frequent personnel turnover endemic to the military, valuable knowledge about files, programs, and models is constantly having to be relearned and, in some cases, recreated. This occurs because knowledge is fragmented and dispersed across several individuals rather than being centralized and accessible to those personnel with a need to know. When an individual leaves the branch, he takes all that personal information with him and his successor must devote considerable time and effort to rediscovering that knowledge base.

This is frequently a problem with programming personnel in a file-oriented environment. Good programmers are difficult to find so that when one appears on the scene, the tendency is to have him write as much code as possible in the time available. Although this may result in more operational programs in the short-term, lack of documentation and the proliferation of files occasioned by these new programs become major problems in the long-term. Further, when the programmer transfers, his replacement is faced with the staggering problem of trying to determine what programs exist, what they do, and what files they use without the benefit of the author to provide guidance. The replacement's startup costs then begin to outweigh the benefits gained from the previous programmer's efficiency.

Again, organizations reach the point where control over information resources becomes as important as the resources themselves. When the lack of a "corporate history" begins to seriously impair the processing efficiency of an organization, then control measures are cost-justified.

2.2 A Two-Phased Strategy for Improvement

DCSPLANS MILPERCEN has reached the stage of information processing maturity where control of information resources is critical. This requires the development of an information resource management approach which can begin to address the problems noted above. In particular, we recommend the following short-term and long-term steps:

1. In the short-term, implement a database administration function within the Special Plans branch whose primary objective is the control and administration of DCSPLANS' data resources.

2. In the long-term, develop an institutional information architecture which is based on the information requirements of the Force Development/Manpower Management process. Current methodologies such as enterprise modeling and information engineering [Martin and Finkelstein 1983] should be used in the development process.

2.2.1 Gaining Control over Data Resources

The short-term phase of this IRM approach involves two related steps: organizational control of data and capturing data about information resources. The second step, in fact, describes one of the critical functions that must be performed once the first step has been undertaken.

2.2.1.1 Organizational control of data

The first recommendation is that Special Plans be formally tasked as the information resource manager of DCSPLANS. This requires the authorization of a data administration cell to establish organizational control over the data and its use.

Implementation of this cell should take place in two steps. Initially, a current military officer from Special Plans should assume this role. This formalizes the role with someone who has the requisite corporate history and who will be recognized as being responsible for, and capable of, handling the necessary data administration.

Secondly, a permanent civilian billet should be established to assist this military officer. This individual will ultimately assume responsibility for DCSPLANS data administration once (s)he becomes familiar with the environment. The responsibilities for this data administrator would include the following:

1. data definition and documentation;

2. database design and implementation;

3. development of, and compliance with, data standards;

4. establishment and enforcement of access, security, and integrity;

5. database software procurement and interface with vendors;

6. liaison, consulting, and training with the user community, i.e. the five branches of DCSPLANS.

A fuller description of the roles and functions of a data administrator can be found in [Leong-Hong and Plagman 1983].

The significant advantages of this approach are at least twofold. First, it establishes a corporate history which is not subject to the vagaries of military personnel turnover. Second, it provides a centralized clearinghouse of information about DCSPLANS resources which, over time, should build confidence about data integrity and reduce the learning curve for people just coming on board.

2.2.1.2 Capturing data about information resources

One of the prime functions of the data administrator is the collection and storage of data about information resources in a **dictionary system**. This dictionary system serves as a centralized knowledge base of relevant DCSPLANS' files, programs, models, and users. Establishment of a dictionary system facilitates answering questions like:

1. Which files and corresponding data elements are input to the P3M model?

2. Who updates the CAUDB and how frequently?

3. What are the allowable ranges of values for certain data elements?

4. What aliases are there for MOS?

5. What program(s) must be run to generate the PMAD and UAD?

This is only a small subset of the possible queries one can direct to a dictionary system. Many other examples are provided in [Kirsch 1985, Noel 1985, Dolk and Kirsch 1986].

The dictionary system is an indispensable tool in the efficient discharging of a data administrator's job. Section 3 takes an in depth look at the structure of a dictionary system and suggests an appropriate model for the DCSPLANS environment.

2.2.2 Developing an Institutional Information Architecture

The short-term strategy described above is, in a sense, a "bottom up" approach to solving DCSPLANS' IRM problems using well-known data management tools and techniques. If properly implemented, it should improve the data integrity situation at the local level of DCSPLANS. The question remains, however, of how well this local solution will integrate with the overall MILPERCEN and U.S. Army information resource environment. Another way of posing the problem is, "at what level of the organization should IRM be implemented?"

The ideal answer to this question would be to do a "top down" approach at as high a level of the organization as possible. There are various methodologies which espouse this approach such as enterprise modeling and information engineering [Martin and Finkelstein 1983]. Information engineering, for example, attempts to capture the information needs of an organization by starting with a definition of the goals of the organization and working down through successive functional layers of detail. Once these needs have been identified, a computerized version of the enterprise is developed based on data management principles and 4th generation languages.

The magnitude of applying information engineering to an organization as large as the U.S. Army, or even MILPERCEN, is overwhelming. Yet, if there is to be a coordinated information environment which satisfies the information needs of an organization, this effort must eventually be made. We recommend, as a long-term project, that information engineering, or some equivalent methodology, be used to identify DCSPLANS' overall information needs. This project should be directed by the data administrator and can make use of the software tools described in Section 3. In this way, the short-term, "bottom up" solution can

merge gracefully with the long-term, "top down" approach.

Until some global attempt is made to identify DCSPLANS' information needs, the problem of data integrity will likely persist. Application of IRM principles is a first step towards gaining control over data resources. Implementation of IRM at the DCSPLANS level should serve as a testbed illuminating the benefits and problems associated with this approach. It should further highlight areas where inconsistencies in information requirements exist between DCSPLANS and external agencies. A successful implementation can serve as the impetus for spreading the IRM philosophy to a wider community within MILPERCEN.

2.3 Summary

The main problems in the DCSPLANS information environment are support of data integrity and lack of a "corporate history" because of military personnel turnover. A short-term solution recommends that a data administrator function be established initially filled by an "on board" officer and then eventually transferred to a civilian individual. The data administrator would be responsible for developing a dictionary system which monitors data integrity (described in the next section). A long-range responsibility of the data administrator would be to conduct a "top down" analysis of DCSPLANS' information needs using some methodology equivalent to information engineering.

3. TECHNOLOGICAL ASPECTS

The establishment of an explicit DCSPLANS data administration function is the first step in gaining control over data resources. The next step is to provide this administrator with the appropriate software tools for implementing this function. We suggest that a dictionary system is the critical software tool required and the control element currently missing from the DCSPLANS environment.

This section describes in detail the structure of a dictionary system which is compatible with emerging Federal standards for dictionaries and shows how it can be used to support DCSPLANS data administration. The material in this section has been summarized from several sources. Sections 3.1, 3.2, and 3.3 are taken from [Dolk and Kirsch 1986]. Prototypes for the dictionary system shown in Section 3.3 have been implemented using Ashton-Tate's dBASEIII micro-DBMS product and are described in [Noel 1985, Kirsch 1985]. The implementation of edit validation rules in a dictionary environment is covered in [DiBona 1985] and implementation of model management via the dictionary is described in [Dolk 1986].

3.1 Dictionary Concepts

Many different terms are used as synonyms for dictionaries but the one most commonly applied is **information resource dictionary system (IRDS)**. An IRDS is essentially a knowledge base about an organization's information resources. It includes capabilities for describing and storing data about information resources as well as retrieving and manipulating this data. Since the data in an IRDS describes other data, it is often referred to as **metadata** and the administration of the dictionary is correspondingly termed **metadata management**.

Dictionary systems typically have two distinct components: the **dictionary** and the **directory**. The dictionary aspect describes **what** information resources exist, what they mean, what their structures are, and how they interrelate. The directory, on the other hand, describes **where** these resources are located and how they are accessed.

Dictionaries are classified as either **passive** or **active**. A passive system does not interact dynamically with any other operational system, i.e. no system depends on the dictionary for its metadata. An active dictionary, on the other hand, generates metadata for one or more processes and is the sole source of that metadata. A database management system (DBMS), for example, may use an active dictionary for all information concerning the description of data items in operational databases. Passive dictionaries are used essentially for documentation and must be updated independently from the operational environment they describe. Active systems are more powerful in implementing control mechanisms but require more overhead in interfacing with other systems. A common implementation strategy is to build a passive system first and then extend it to an active one for selected applications. Techniques for doing this in the DCSPLANS environment will be discussed later in this section.

IRDS are also characterized as either **DBMS-dependent** or **free-standing (DBMS-independent)**. A DBMS-dependent IRDS needs an underlying DBMS to perform metadata retrieval and manipulation. A freestanding IRDS supplies all those functions internally. A DBMS-dependent IRDS can be be built from scratch more readily but is constrained to operate in an environment containing the underlying DBMS. A free-standing IRDS is more versatile in this regard yet more costly to build.

In summary, the active/passive designation refers to whether or not other operational systems need the IRDS for their metadata whereas the DBMS-dependent/independent classification refers to whether or not the IRDS needs a DBMS to perform its manipulation functions. We recommend that DCSPLANS first build a passive, DBMS-dependent IRDS and then convert it to an active, dependent system. Details for accomplishing this are presented in the remainder of this section.

3.2 FIPS IRDS

It has been estimated that the Federal government can realize $120 million in benefits by the early 1990's from the use of a standard IRDS. As a result, the National Bureau of Standards has developed specifications for an IRDS which will form the basis for a Federal Information Processing Standard (FIPS) IRDS. These specifications include many of the functions available in existing commercial dictionary systems while also providing flexibility for tailoring the IRDS to specific information administration requirements.

The central feature of the FIPS IRDS is the **core system-standard schema (core)** which describes the logical structure of the IRDS itself. The core consists of entity, attribute, and relationship types as shown in Appendix A. Entity types correspond to the various objects which exist in an IRM environment such as files, programs, and users. Attribute types are simply descriptors of entity types. The core supports three distinct name attributes for each entity: ACCESS-NAME, DESCRIPTIVE-NAME, and ALTERNATE-NAME. ACCESS-NAME is a short, easy to use, and unique name with which the user will most frequently interact whereas DESCRIPTIVE-NAME provides a more meaningful, but also unique, name. ALTERNATE-NAME allows multiple aliases to be associated with any one entity. Relationship types capture the important associations between entities that exist in an information resource environment. An important feature of the FIPS IRDS core is that all relationships between entity types are binary in nature. Further, there are constraints as to which entity types are allowed to participate in which relationships (see Table A-4 in Appendix A). For example **PROCESSES(system,file)** is legal but not **PROCESSES(file,system)** since a file cannot process a system.

The intent of the FIPS IRDS specifications is that the core serve as a common baseline from which to implement IRM. It's not expected that the core will be sufficiently robust to support all IRM environments, however. As a result, the core is characterized as being **extensible** in that additional entity, attribute, and relationship types can be added to support unique requirements. Thus, each installation has the flexibility to tailor the IRDS to their specific information resource environment.

3.3 Relational Model of FIPS IRDS (RIRDS)

The FIPS specifications mandate that an IRDS implementation will be considered in compliance if it supports fully the core and additionally supports either a panel-driven (menu-driven) or command language interface. A DBMS-dependent IRDS is a logical way to incorporate a command language interface since DBMS's automatically provide data sublanguages with which to describe, manipulate, and control data. In particular, a relational DBMS (RDBMS) based on the SQL data sublanguage provides an ideal environment for implementing the FIPS IRDS. This section

describes such an implementation performed with the relational ORACLE DBMS developed by the ORACLE Corporation. No claims are made for the relative superiority of ORACLE vis-a-vis other systems. In fact, the implementation described herein should be easily transportable to other DBMS environments. Familiarity with SQL is assumed in the following discussion. See Appendix B for a concise summary of SQL.

A simple relational model of the core types is shown in Figure 3-1. Notice that the entity types have many attibutes in common but that some attributes (e.g.: lines-code) are only associated with a subset of the entity types as shown in Table A-2. This relational version of the core can be simplified into 2 relations, ENTITY and RELSHIP (Figure 3-2a) by using the relational view mechanism as embodied in the SQL language. For example, we can impose the view PROGRAM (as shown in Figure 3-1) on ENTITY with the following SQL command:

```
CREATE VIEW PROGRAM AS
(SELECT ANAME,DNAME,ADDED_BY,DATE_ADDED,MOD_BY,LAST_MOD,
        NMODS,DUR_VALUE,DUR_TYPE,LINES_CODE,COMMENTS,SECURITY
 FROM   ENTITY
 WHERE  ETYPE='PROGRAM');
```

Similarly, we can impose the view PROCESSES (as shown in Figure 3-1) on RELSHIP:

```
CREATE VIEW PROCESSES AS
(SELECT E1NAME, E1TYPE, E2NAME, E2TYPE
 FROM   RELSHIP
 WHERE  RTYPE='PROCESSES');
```

Notice that the attribute **etype** in ENTITY must be one of the entity types shown in Figure 3-1 and **rtype** in RELSHIP must be one of the relationship types. By creating a view for each entity type and relationship type in the FIPS IRDS, we create the logical equivalent of Figure 3-1 using only two underlying relations.

3.3.1 Self-Descriptive IRDS

Since an IRDS describes information resources, it should be able to describe itself. This implies that the information resource administrator should be able to determine from the IRDS which entity, attribute, and relationship types the IRDS supports as well as the relationship constraints in effect (see Table A-4). Self-descriptive capabilities facilitate a strong integrity checking mechanism as we show later. The relational model described so far has very limited self-descriptive features. Although we could determine which entity and relationship types exist by the following two commands:

```
SELECT UNIQUE ETYPE FROM ENTITY;

SELECT UNIQUE RTYPE FROM RELSHIP;
```

Entities and Attributes

SYSTEM(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,comments,descr,security)

PROGRAM(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,lang,lines-code,comments,
descr,security)

MODULE(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,lines-code,comments,descr,
security)

FILE(aname,dname,added-by,date-added,mod-by,last-mod,nmods,
nrecs,comments,descr,security)

RECORD(aname,dname,added-by,date-added,mod-by,last-mod,nmods,
rec-cat,comments,descr,security)

ELEMENT(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,data-class,low-range,high-range,comments,
descr,security)

DOCUMENT(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,doc-cat,comments,descr,security)

USER(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,comments,descr,location,security)

Relationships

All relationships have the same attributes and keys:

REL(e1name,e1type,e2name,e2type)

where e1name,e2name are the entity instances

e1type,e2type are the entity-types of which e1name,
e2name are instances, respectively

REL is any of the relationships CONTAINS, PROCESSES,
RUNS, RESP_FOR, CALLS, GOES_TO, DERIVED_FROM,
ALIAS and KWIC.

Integrity Constraints

(Please see Table A-4.)

Figure 3-1: Relational IRDS (RIRDS)

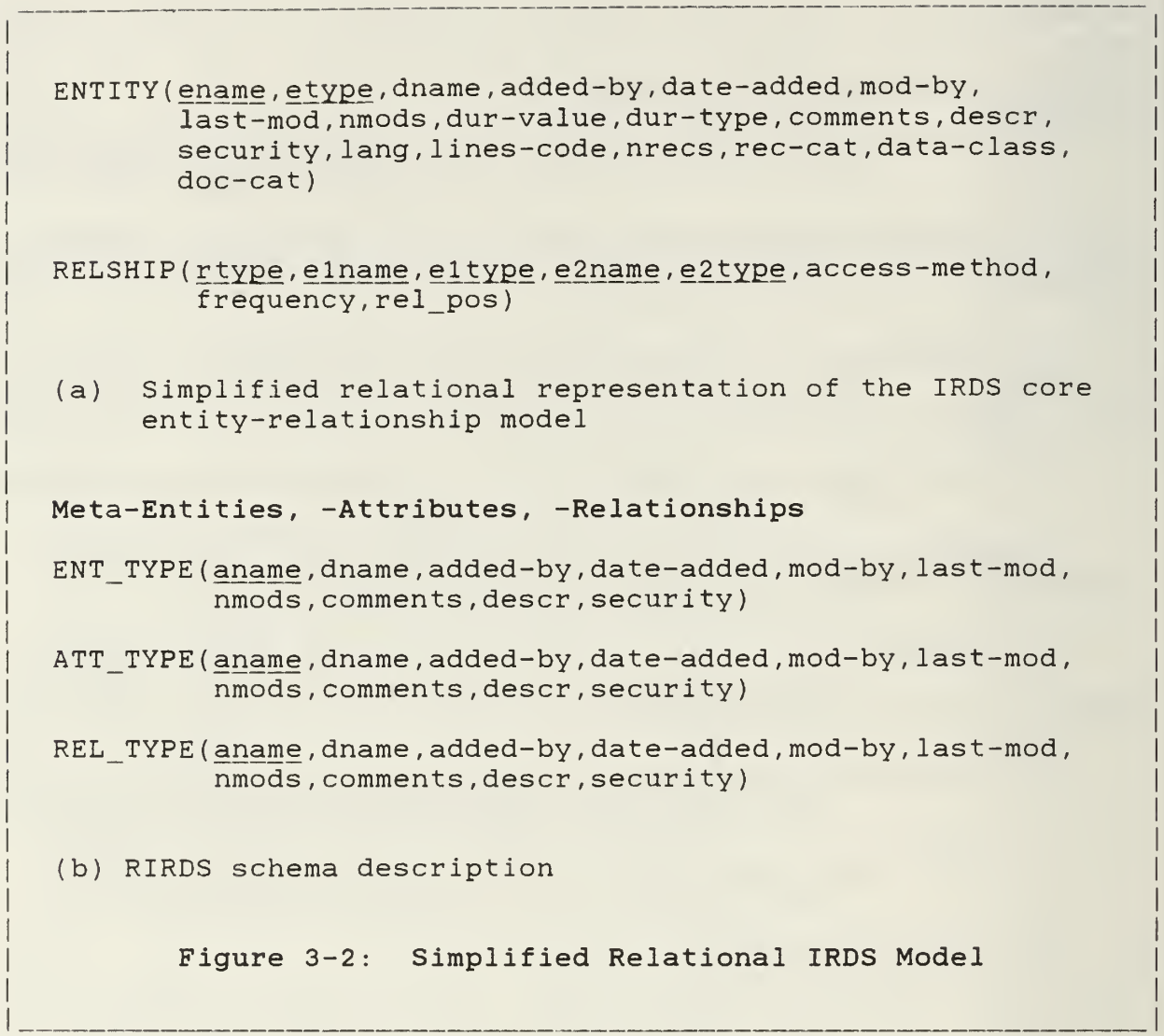


Figure 3-2: Simplified Relational IRDS Model

this is far from ideal. For one thing, it's possible that the IRDS may support an entity type (e.g.: MODULE) for which no instances have been entered yet. In this case, the first query would not show tha MODULE was an entity type. A similar situation holds for relationship types in the second query. Further, the relational model in its current form has no way of describing the relationship constraints.

In order to make the IRDS self-descriptive, three new relations must be added corresponding to each of the types: ENT_TYPE, ATT_TYPE, and REL_TYPE (Figure 3-2b). These meta-relations describe relations or views existing at the ENTITY/RELSHIP level. For example, the domain of **ENTITY.etype** is defined by the set of values of **ENT_TYPE.aname**; similarly for **RELSHIP.rtype** and **REL_TYPE.aname**. Now the entity and relationship types can be listed independent of whether actual instances of these types are in the database:

```
SELECT ANAME FROM ENT_TYPE;

SELECT ANAME FROM REL_TYPE;
```

Further, the relationship constraints can now be represented explicitly in the IRDS as instances in the appropriate relationship view. For example to represent the constraint PROCESSES(system,file) requires an entry in RELSHIP as follows:

```
RELSHIP('processes', 'system', 'ent_type', 'file', 'ent_type')
```

Once the constraints have been entered, the administrator can retrieve information about the IRDS itself. For example, to determine which relationships the entity type PROGRAM can legally participate in, the administrator would issue the following SQL command:

```
SELECT RTYPE, E1NAME, E2NAME FROM RELSHIP
WHERE E1NAME='program' OR E2NAME='program'
```

3.3.2 Compatibility with the FIPS IRDS

The RIRDS deviates from the FIPS IRDS core in two respects: physical representation and multiple attributes. The entity types BIT-STRING, CHARACTER-STRING, FIXED-POINT, and FLOAT have been omitted as well as the relationship type REPRESENTED-AS. These types are concerned with the physical representation of data (ELEMENT entities) whereas the rest of the core is concerned with logical relationships. Further the FIPS IRDS approach in this case precludes many realistic situations wherein an element entity (e.g., SOCIAL_SECURITY_NUMBER) may appear as a FIXED-POINT in one file and CHARACTER-STRING in another file. We recommend inbstead embedding this information as an attribute type (e.g., FORMAT) in the FILE-CONTAINS-ELEMENT relationship type.

Multiple attribute types have been omitted from the RIRDS since these violate first normal form. Multiple attributes are those which may have more than one occurrence for each entity. For example, the attribute type LOCATION may have several values for a file which is distributed at various nodes in a network. Multiple attributes require new relations to be defined in addition to the two shown in Figure 3-2a and thus complicate the model. In the cases where these attributes are vital (e.g.: ALTERNATE-NAME and CLASSIFICATION), new relationship types have been defined (ALIAS and KWIC, respectively) to accommodate the situation. ALIAS provides a valuable synonym capability which allows multiple names to be assigned to the same entity. It's defined by the following SQL command:

```
CREATE VIEW ALIAS AS
(SELECT E1NAME, E1TYPE, E2NAME FROM RELSHIP
 WHERE RTYPE = 'alias')
```

The key-word-in-context (KWIC) allows entities to be classified according to user-chosen categories and facilitates queries of

the kind, "list all entities associated with REENLISTMENT". It's defined analogously to ALIAS:

```
CREATE VIEW KWIC AS
(SELECT E1NAME, E1TYPE, E2NAME FROM RELSHIP
 WHERE RTYPE = 'kwic')
```

Other multiple attributes which the data administrator considers vital can be included using the same kind of strategy.

3.4 Uses of IRDS in DCSPLANS

This section discusses how an IRDS can be used within the DCSPLANS environment to determine and maintain data integrity, support model management, and monitor access to information resources.

3.4.1 Data Integrity

Section 2 identified data integrity as one of the critical problems in DCSPLANS' information processing. The IRDS affords several mechanisms for bringing this problem under control: built-in consistency checking, file update monitoring, and support for edit and validation rules. Each of these is discussed in detail below.

3.4.1.1 Consistency checking

The self-descriptive capability of the dictionary allows the IRDS to check whether its contents are consistent with its own logical description. For example, someone may inadvertently have entered the information PROCESSES('pos_edit', 'file', '????', 'program') which violates the acceptable constraints for PROCESSES as shown in Table A-4 since a file cannot process a program. The following SQL command identifies all violations of PROCESSES constraints ("!" is equivalent to "NOT"):

```
SELECT * FROM PROCESSES
WHERE E1TYPE != 'ent_type' AND E2TYPE != 'ent_type' AND
     (E1TYPE, E2TYPE) NOT IN
      (SELECT E1NAME, E2NAME FROM PROCESSES
       WHERE E1TYPE = 'ent_type' AND E2TYPE = 'ent_type')
```

The way this query works is that the subquery (2nd SELECT clause) retrieves the set of all pairs of entity-types which may legally participate in PROCESSES. The first SELECT clause then identifies and displays any pairs of entities in the meta-data appearing in PROCESSES whose entity types do not fall in that set. All invalid occurrences can subsequently be deleted from the database by simply changing "SELECT *" to "DELETE" in the above query.

Notice that the above query can be used for any relationship type by simply changing "PROCESSES" to the appropriate relationship

type name. A global consistency check can be performed by replacing "PROCESSES" with "RELSHIP". This would check all constraints in Table A-4 for possible violations, thus one SQL command is all that's required to determine the integrity of the data in the IRDS itself.

The power of SQL in integrity checking is somewhat offset by the complexity of the required commands. This can be neatly circumvented, however, by taking advantage of macro facilities which most relational DBMS provide. In the ORACLE system, for example, the global consistency query might be saved as the macro GLOBAL_CHECK which could then be invoked directly without having to know the complexities of SQL. One of the duties of the data administrator is to define an appropriate set of such macros for the user commmunity (see Appendix C for a representative sample).

3.4.1.2 File update monitoring

One of the ongoing problems at DCSPLANS has been monitoring the update activity of files which serve as the source of input data to various manpower models. The IRDS provides built-in features for tracking file update activity via the LAST-MODIFICATION-DATE and LAST-MODIFIED-BY attribute types (**last-mod** and **mod**-by respectively in the RIRDS (Figure 3-2)). For example, the following SQL command will display modification information for all files processed by the P3M model:

```
SELECT ENAME, MOD-BY, LAST-MOD FROM FILE, PROCESSES
WHERE E1NAME = 'p3m' AND E1TYPE = 'program' AND
      E2TYPE = 'file' AND E2NAME = ENAME
```

The above query requires that current information be maintained in the IRDS about file update activity. This may be done in two ways: using the IRDS in passive mode or using the IRDS in active mode. A passive IRDS requires that someone enter all modification information explicitly into the dictionary. This assumes that all such information is available in the first place. Even if this were true, the data entry task would be formidable.

A preferable approach is to make the IRDS serve as a control tool in the active mode. This would require all file update transactions to be routed through the IRDS so that the appropriate information could be logged. The easiest way to achieve this would be to write a file access program (FAP) which accesses the IRDS and which all users must invoke as part of the process of modifying a file (Figure 3-3). FAP would prompt the user for the runstream(s) (JCL file(s)) to be executed, schedule the job(s) to be run, and finally update the appropriate IRDS modification information. This could be accomplished by adding RUNSTREAM as an entity type to the IRDS, and then, for each runstream entity in the IRDS, entering the appropriate entities in the PROCESSES(runstream, file) and PROCESSES(runstream, element) relationships. In this way, file update information could be logged automatically in the IRDS while simultaneously

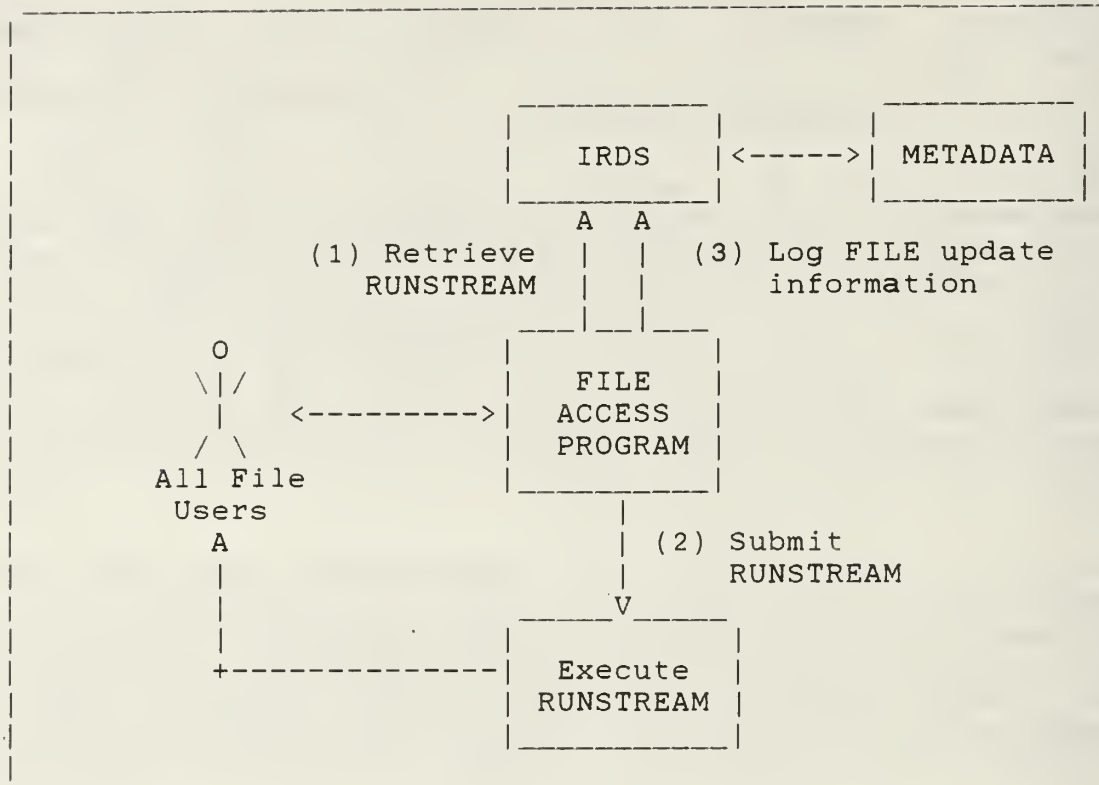


Figure 3-3: Active IRDS to Monitor File Update Activity

implementing an important control valve previously lacking in the DCSPLANS environment.

There is one limitation in the scenario sketched above and that is only the most recent file (or element) update transaction is recorded in the IRDS. If it were desirable to maintain a log or history of file (or element) updates, the IRDS structure would have to be modified in order to accommodate the time dimension. In particular, the attribute type **last-mod** would have to become part of the key in FILE (or ELEMENT) where **last-mod** would serve as a timestamp to differentiate multiple entries for a particular file or element.

Although adding timestamps is not a difficult change to make conceptually, one must be alert to the potential for rapid growth in the size of the IRDS, particularly in a high update environment. Careful attention must be paid in this situation to eliminating or archiving historical information not of immediate relevance. The following SQL command, for example, would eliminate all historical information about the P3M file previous to 01 July 86 :

```
DELETE FROM FILE
WHERE LAST-MOD < '86-JULY-01' AND ENAME = 'p3m'
```

The extensibility of the IRDS plus its ability to handle rudimentary time semantics provides a file access control mechanism which can contribute to DCSPLANS' quest for improved data integrity. Note, however, that this must be implemented in conjunction with a policy, stated and enforced, which mandates that all file update activity must take place via FAP. (Details of the FAP implementation depend upon the DBMS employed for the RIRDS and are beyond the scope of this report. See Section 3.5, however, for related information.)

3.4.1.3 Edit and validation rules (EVR)

DiBona's thesis [DiBona 1985] discusses the use of an active IRDS to implement edit and validation rules (EVR). EVR's are constraints which define valid data values. The constraints shown in Table A-4 are examples of EVR's for the IRDS.

EVR's generally fall into one of three constraint categories: field, intra-record, or inter-record. Field constraints usually specify a range of values within which the field value must fall (e.g., the grade value must fall within E1 to E9). Intra-record and inter-record constraints specify field values which depend on the values of one or more fields in the same, or different records respectively (e.g., if the POS code in a record is 63H, then the grade value in that record must be E4 or E5).

Ideally, we would like to be able to store the required EVR's in the IRDS and then trigger them on the actual data in a fashion similar to that shown in Section 3.4.1.1. Although lower and upper ranges of field constraints can be represented using the ALLOWABLE-RANGE attribute (see Table A-2), the intra- and inter-record constraints cannot be implemented straightforwardly because there is no explicit way of representing the "if..then.." form of the rules in the IRDS.

As a result, the enforcement of EVR's is better left to a separate program which works from the dictionary metadata. Figure 3-4 shows how such a data filter system might be configured. The question marks shown on the link between the dictionary and the EVR's reflects the fact that the representation of the intra- and inter-record rules in the dictionary is still an unresolved issue. Currently, those rules must be embedded within the EVR program itself.

One possible way around this dilemma of representing integrity constraints might be to create a new entity-type CONSTRAINT which contains the SQL representation of the constraint if it were violated. As an example, consider the intra-record constraint used above. The SQL command for identifying violations of the constraint would be (assume we're using file POS_FILE with the key MID for military-id):

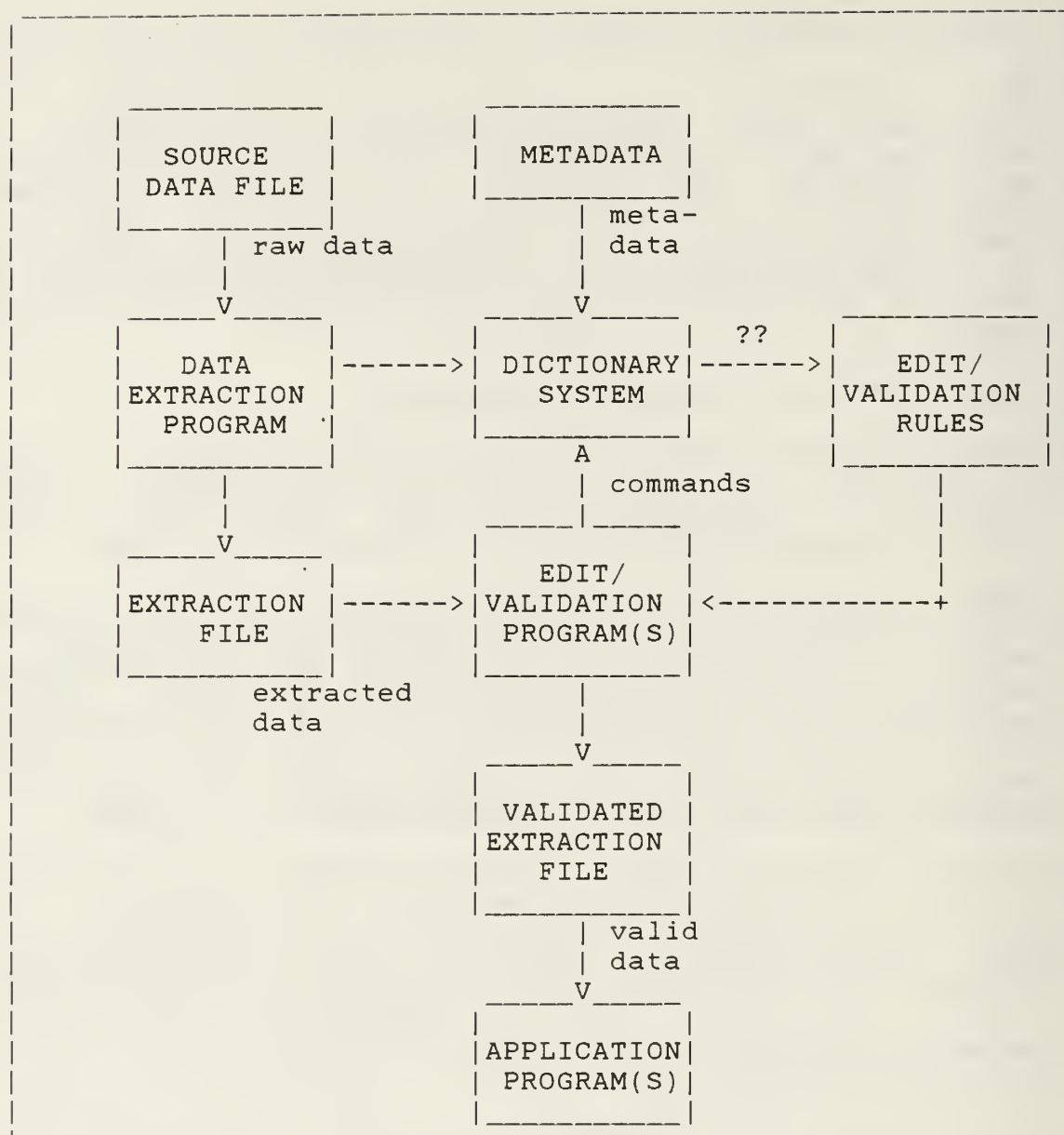


Figure 3-4: EVR Data Filter System

```
SELECT MID,POS,GRADE FROM POS_FILE
WHERE POS = '63H' AND GRADE != 'E4' AND GRADE != 'E5'
```

A relationship-type ATTRIBUTE-HAS-CONSTRAINT would also have to be created to associate attributes with their corresponding constraint(s). The EVR program could then retrieve these commands from the IRDS and apply them to the actual POS_FILE itself. In this case only violations of the integrity constraint would be displayed. This would require, however, that the POS_FILE be stored in a relational database with the same SQL

capability. Further, stating a constraint in the negative is confusing to someone who is not familiar with Boolean logic and its manipulation.

The problem of representing EVR's in the IRDS requires further investigation. The ideal solution would be to extend SQL to a recursive, rule-based language like Prolog but currently no such languages exist which interface with industrial strength relational database systems.

3.4.2 Model Management

DCSPLANS builds and maintains many sophisticated manpower forecasting models in support of its primary function. As a result, DCSPLANS must be concerned with model management, i.e. the management of models as an organizational resource. This entails an integrated, coherent approach to building, maniplating, and analyzing models as well as communicating the results of models to relevant decision-makers. The underlying concept is that models are a sharable resource in the same way that data are.

Until recently model management has been more of a "buzzword" than a reality but that is rapidly beginning to change. With the introduction of structured modeling [Geoffrion 1985, 1986], model management has a theoretical framework comparable in power to the relational model which has fueled the data management movement since the early 70's.

This section summarizes the main benefits of the structured modeling approach and sketches how structured modeling can be implemented via the IRDS. A more detailed description of these topics is provided in [Dolk 1986].

3.4.2.1 Structured modeling

Structured modeling facilitates and integrates many aspects of the modeling process.

> "Structured modeling endeavors to provide a formal mathematical framework, language, and computer-based environment for conceiving, representing, and manipulating a wide variety of models." [Geoffrion 1986]

The basic motivation for structured modeling arose from the long standing problems which operations research and management science practitioners have faced in "selling" their models to management. The OR/MS community suffers from an identity crisis in this regard whereas management, on the other hand, tends to regard modeling people as technophiles with little understanding of the "real world" decision-making environment. As a result, there is an acute need for increased communication between the modeling community and management.

Much of this communication gap is due to the cumbersome modeling systems which have traditionally been used in building and analyzing models. A modeling system which addresses the current crisis must have the following characteristics:

1. A conceptual framework defining a single model representation;

2. Independence of the model representation from model solution operators and from the underlying data associated with the model;

3. The ability to represent a wide range of OR/MS mathematical models;

4. Support for the overall modeling life cycle;

5. Full use of data management facilities such as RDBMS and RIRDS.

Structured modeling provides a framework which satisfies all of the above criteria. Additionally, it offers the following benefits:

1. Top-down model design: mathematical models can be built top-down in the same way that computer software is built using structured programming techniques.

2. Integrated modeling: models can also be built bottom-up or middle-out by linking two or more models into a composite (and more complex) model; this is very difficult to accomplish with current technology because modeling systems don't interface with other modeling systems in any significant fashion.

3. Communication and documentation: models can be represented graphically or as schemas with different views; mathematical and natural language representations of these schemas can be generated depending upon the audience.

Structured modeling is a comprehensive and integrated approach to modeling which is ideal for model management purposes and consistent with information resource management practices. Appendix D provides a brief summary of the fundamental concepts used in structured modeling. The next section sketches how the IRDS can accommodate structured model representations.

3.4.2.2 A model management system based on structured modeling

In the same way that a DBMS provides a tool for implementing data management, a model management system (MMS) is required to support model management. An MMS must provide model description, manipulation, and control functions and must support the sharing of models and their underlying data. A preliminary version of an MMS based on structured modeling can be implemented by a simple

Entity-Types

```
ENT_TYPE('pe', 'primitive_entity', ....)
ENT_TYPE('ce', 'compound_entity', ....)
ENT_TYPE('att', 'attribute', ....)
ENT_TYPE('va', 'variable_attribute', ....)
ENT_TYPE('test', 'test_entity', ....)
ENT_TYPE('fcn', 'function_entity', ....)
ENT_TYPE('model', 'model', ....)
```

Entities

```
PE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

CE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

ATT(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

VA(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

TEST(aname, dname, .... , doc_cat, index,
     index_stmt, gen_range, gen_rule)

FCN(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)

MODEL(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)
```

Integrity Constraints

```
CALLS(ce,pe)        CALLS(va,pe)        CALLS(test,test)
CALLS(att,pe)       CALLS(va,ce)        CALLS(test,fcn)
CALLS(att,ce)       CALLS(test,va)      CALLS(fcn,fcn)
CALLS(test,att)     CALLS(fcn,va)       CALLS(fcn,test)
CALLS(fcn,att)

CONTAINS(module,module)          CONTAINS(module,test)
CONTAINS(module,pe)              CONTAINS(module,fcn)
CONTAINS(module,ce)              CONTAINS(model,module)
```

Figure 3-5: IRDS Representation of Structured Modeling

extension of the IRDS described above. Familiarity with structured modeling terminology as described in Appendix D is assumed.

In order to accommodate the representation of structured models, the IRDS core must be extended to include new entity types corresponding to the genus types (primitive entity (pe), compound entity (ce), attribute (att), variable attribute (va), test (test), and function (fcn)) in structured modeling (Figure 3-5). The entity type **genus** is also a useful, although not strictly necessary, addition. The entity type **model** should also be added to reflect models as an important resource. These entity types will then be implemented as entities by establishing the appropriate views on the ENTITY relation. Constraints governing generic structure (i.e., acceptable calling sequences) and modular structure must also be defined in the IRDS.

With this extended IRDS core, it is now possible to represent structured models in relational form. The IRDS representation of the transportation model used as an example in Appendix D is shown in Figure 3-6.

This representation facilitates several different kinds of queries. Appendix C enumerates some model validity commands which could be established by the data/model administrator. Calling sequences for a particular genus (e.g.: FLOW) can be determined via the following SQL command:

```
SELECT E2NAME, E2TYPE FROM CALLS
WHERE E1NAME = 'flow'
```

A natural language summary of the transportation model for managers and a mathematical summary for modelers can be generated by the following commands respectively:

```
SELECT ENAME, ETYPE, INDEX, COMMENTS
FROM GENUS

SELECT ENAME, ETYPE, INDEX, INDEX_STMT
FROM GENUS
```

Numerous other retrievals can be made to support either modeling or administration functions.

One of the powerful features of structured modeling is that the model schema automatically defines a relational form for the underlying data (or elemental detail) associated with the model. The generic structure defines functional dependencies between model components and these dependencies can be transformed automatically into a set of relations (see [Geoffrion 1985] for one approach to this transformation) defining the elemental detail of the model. Figure 3-7 shows the elemental detail relations for the transportation model.

Entities

```
MODEL('TRANSP', 'Transportation_Model', ....)
MODULE('&PROD', 'Source_Data', ....)
MODULE('&SALES', 'Customer_Data', ....)
MODULE('&DIST', 'Transportation_Data', ....)
PE('PLANT', 'Mfg_Plant', ....)
PE('CUST', 'Customer', ....)
CE('LINK', 'Link_Plant_&_Customer', ....)
ATT_TYPE('SUP', 'Plant_Supply_Capacity', ....)
ATT_TYPE('DEM', 'Customer_Demand', ....)
ATT_TYPE('FLOW', 'Transportation_Flow', ....)
ATT_TYPE('COST', 'Transportation_Cost', ....)
TEST('T:SUP', 'SUMj(FLOWij) <= SUPi', ....)
TEST('T:DEM', 'SUMi(FLOWij) = DEMj', ....)
FCN('TOTAL_COST', 'SUMij(COSTij * FLOWij)', ....)
```

Generic Structure

```
CALLS('SUP', 'ATT', 'PLANT', 'PE')
CALLS('T:SUP', 'TEST', 'SUP', 'ATT')
CALLS('DEM', 'ATT', 'CUST', 'PE')
CALLS('T:DEM', 'TEST', 'DEM', 'ATT')
CALLS('LINK', 'CE', 'PLANT', 'PE')
CALLS('LINK', 'CE', 'CUST', 'PE')
CALLS('COST', 'ATT', 'LINK', 'CE')
CALLS('FLOW', 'VA', 'LINK', 'CE')
CALLS('T:SUP', 'TEST', 'FLOW', 'VA')
CALLS('T:DEM', 'TEST', 'FLOW', 'VA')
CALLS('TOTAL_COST', 'FCN', 'FLOW', 'VA')
CALLS('TOTAL_COST', 'FCN', 'COST', 'ATT')
```

Modular Structure

```
CONTAINS('TRANSP','MODEL','&PROD','MODULE')
CONTAINS('TRANSP','MODEL','&SALES','MODULE')
CONTAINS('TRANSP','MODEL','&DIST','MODULE')
CONTAINS('TRANSP','MODEL','TOTAL_COST','FCN')
CONTAINS('TRANSP','MODEL','T:SUP','TEST')
CONTAINS('TRANSP','MODEL','T:DEM','TEST')
CONTAINS('&PROD','MODULE','PLANT','PE')
CONTAINS('&PROD','MODULE','SUP','ATT')
CONTAINS('&SALES','MODULE','CUST','PE')
CONTAINS('&SALES','MODULE','DEM','ATT')
CONTAINS('&DIST','MODULE','LINK','CE')
CONTAINS('&DIST','MODULE','COST','ATT')
CONTAINS('&DIST','MODULE','FLOW','VA')
```

Figure 3-6: IRDS Representation of Transportation Model

PLANT(plant_id, supply, test:sup, interpretation)

CUST(cust_id, demand, test:dem, interpretation)

LINK(plant_id, cust_id, cost, flow, interpretation)

Figure 3-7: Relational Form of Elemental Detail for Transportation Model

In order to fully support the elemental detail aspect of structured modeling, an external program must be written to perform the transformation from generic structure to the appropriate relations. There is no SQL command or combination of commands which can perform this operation.

Perhaps the most appealing aspect of structured modeling is this coordination of models and data. Defining the model automatically defines the data requirements as a by-product. The data can then be queried in conjunction with the model schema and vice versa. This provides a high degree of flexibility not found in most modeling systems.

3.4.2.3 Model management in DCSPLANS

Because of DCSPLANS' model-oriented mission, serious consideration should be given to adopting some form of model management. Structured modeling seems to provide the necessary ingredients for a powerful and integrated model management system. It is a new concept, however, which requires further experimentation and refinement. No functional model management system based on structured modeling yet exists although a prototype is currently under development using the ORACLE RDBMS [Dolk 1986].

A sensible migration strategy would be to first implement the RIRDS to gain more control over data integrity. Once the IRDS becomes institutionalized, then it can be expanded in scope to accommodate the model management activities of DCSPLANS. The next section discusses in more detail some of the issues related to implementation.

3.5 RIRDS Implementation Considerations

Implementing a dictionary system like the RIRDS can be a complex and, at times, frustrating experience. This section provides some guidelines concerning the implementation of the RIRDS in DCSPLANS.

3.5.1 Selective Retrofitting

Implementing an IRM environment requires careful planning. There are basically two general approaches which can be invoked: top-down and bottom-up. Top-down usually involves a high level organizational requirements analysis using techniques like information engineering or enterprise modeling. This implies a complete overhaul of the existing data processing environment and is probably not appropriate for the DCSPLANS situation since such an effort would involve external agencies and require high level MILPERCEN coordination..

Bottom-up is more evolutionary and phases IRM into the organizational framework via a process called **selective retrofitting**. Attempting to catalog the overall information resource environment in the IRDS is equivalent in scope and effort to performing a complete requirements analysis. Many dictionary implementations have floundered or failed precisely for this reason. Selective retrofitting involves identifying the critical data and applications which need to be cataloged and initially restricting the contents of the IRDS to these. Once the dictionary has been built, new applications and new data resources are incorporated into the IRDS as they emerge. As a result the IRDS evolves over a period of time to capture more and more of the information resource environment. This gradual process requires significantly less startup cost and effort and provides more flexibility in tailoring the IRDS to dynamic situations.

Selective retrofitting is definitely recommended as the preferable approach for DCSPLANS. DCSPLANS should prioritize which data, programs, models, etc. are to be cataloged in the IRDS. Programs or models which are apt to change in the near future should be assigned low priority. As new applications arise, they can be assimilated according to established IRM policy.

3.5.2 Implementing the RIRDS Using the ORACLE System

The RIRDS was designed based on the availability of an SQL-compatible RDBMS. A partial description of an actual implementation using the ORACLE RDBMS is presented in [Dolk and Kirsch 1986]. The following steps should be followed in building the IRDS independent of whether a relation system is used (see next section for a network model of the IRDS). The corresponding ORACLE commands are provided in parentheses.

1. Define the entity, attribute, and relationship types to be included in the IRDS. These should include the core types as well as extensible types necessary for the particular environment. e.g.: RUNSTREAM in DCSPLANS.

2. Define the appropriate integrity constraints, i.e. which entity types may participate in which relationship types (see Figure A-4).

3. Build the 5 relations shown in Figure 3-2 (using the CREATE TABLE command).

4. Establish the entity types as the appropriate views on ENTITY and the relationship types as the appropriate views on RELSHIP (using the CREATE VIEW command).

5. Enter the entity, attribute, and relationship types which will appear in the IRDS as tuples in the ENT_TYPE, ATT_TYPE, and REL_TYPE relations respectively (using the INSERT INTO table VALUES command). For example,

```
INSERT INTO ENT_TYPE VALUES
('system', 'information_system', .... )
```

6. Show which attribute types are associated with which entity and relationship types via entries in the CONTAINS relationship (using the INSERT INTO CONTAINS VALUES command). For example,

```
INSERT INTO CONTAINS VALUES
('program', 'ent_type', 'lines-code', 'att_type')
```

7. Represent the constraints from step 2. as tuples in the appropriate relationships (using the INSERT INTO table VALUES command). For example,

```
INSERT INTO PROCESSES VALUES
('system', 'ent_type', 'file', 'ent_type')
```

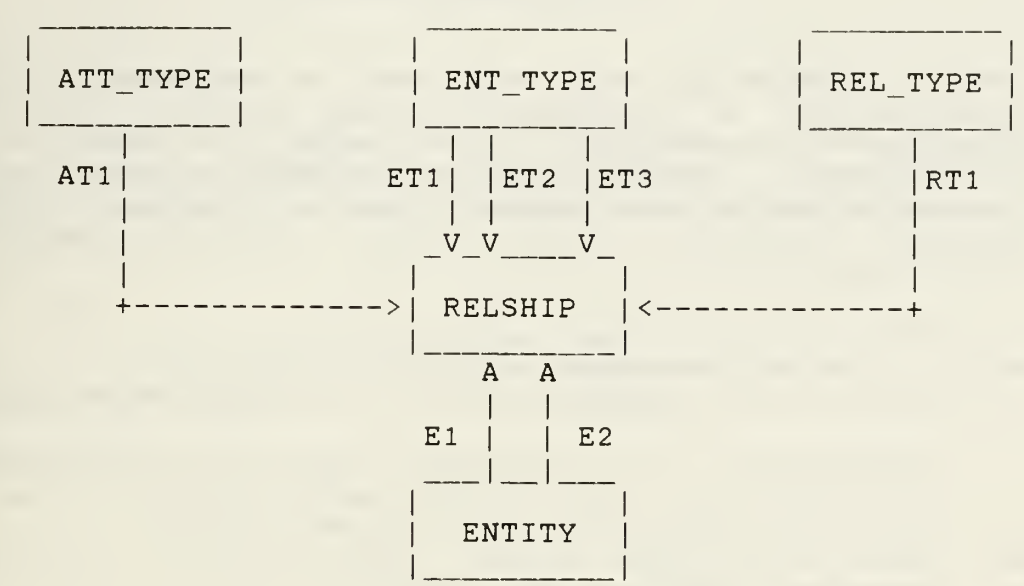
8. Steps 1-7 comprises the IRDS schema information which makes the IRDS self-descriptive. Once this information is entered, then the regular metadata can be added to the dictionary, for example,

```
INSERT INTO FILE VALUES
('pos_edit', 'pos_edit_file', ... )
```

New entity, attribute, and relationship types can be added to the core as the need arises. This situation is shown in Table 10 of [Dolk and Kirsch 1986].

3.5.3 Network Implementation of IRDS

It is strongly recommended that a relational DBMS be used as the basis of implementing the IRDS because of the logical flexibility of the relational environment. This may not always be feasible, however, so we indicate here a network model of the IRDS which may be more suitable for hierarchical or CODASYL DBMS's such as S2000. Since no implementation has been attempted, we only sketch a possible configuration (Figure 3-8) with the understanding that other designs may be more appropriate.



Sets	Set Description
AT1	ATTTYPE-CONTAINED-IN
ET1	ENTTYPE-CONTAINS-ATTYPE
ET2	ENTTYPE-REL-ENTTYPE
ET3	ENTTYPE-RELEDBY-ENTTYPE
RT1	RELTYPE-CONTAINS-ATTTYPE
E1	ENTITY-REL-ENTITY
E2	ENTITY-RELEDBY-ENTITY

Figure 3-8: A Network Design for the IRDS

4. CONCLUSIONS

IRM is becoming more and more a necessary ingredient in organizations as management becomes aware of the benefits of planning, controlling, and using information resources intelligently. Our objective has been to show how IRM can be gradually assimilated into DCSPLANS with minimal disruption of ongoing activities. Specifically, we recommend the following three courses of action:

1. Institution of a data administration billet;

2. Adoption and implementation of an active FIPS IRDS;

3. Consideration of structured modeling as a discipline for coordinating and facilitating modeling activities.

4.1. Permanent Civilian DBA Billet

IRM cannot be treated as an afterthought which is assigned to personnel as something to do after their primary duties are discharged. It is a full-time job which requires dedicated personnel and financial resources if it is to work successfully. DCSPLANS is at the stage of growth where a data administration billet is required to handle the problems of data integrity and lack of "corporate history". We recommend that such a billet be instituted for DCSPLANS use according to the guidelines set forth in Section 2.

4.2. Adoption of FIPS IRDS

An IRDS is an indispensable tool for effective IRM. We have presented a design for such a system which is compatible with emerging Federal standards and which can be used to ameliorate many of the data integrity problems which DCSPLANS currently suffers. We recommend that some version of the IRDS, preferably relational, be implemented to support the data administrator. Further the IRDS should be activated so that it can serve as a control mechanism for monitoring access to existing information resources.

4.3. Adoption of Structured Modeling

Because of the intensive modeling-oriented nature of DCSPLAN activities, consideration should be given to adopting structured modeling as a way of unifying, describing, manipulating, and documenting models. As we have shown, structured modeling is consistent with IRM practices and would further serve to coordinate DCSPLANS information environment.

5. REFERENCES

[Broome 1985]

Broome, R.E. An analysis of information resource management within the Deputy Chief of Staff for Plans, U.S. Army Military Personnel Center. Naval Postgraduate School Master's Thesis, March 1985.

[Date 1982]

Date, C.J. **An Introduction to Database Systems**, 3rd edition, Addison-Wesley, 1982.

[DiBona 1985]

DiBona, R. Use and design of an active data dictionary for local validation of input data. Naval Postgraduate School Master's Thesis, March 1985.

[Dolk 1986]

Dolk, D.R. Model management and structured modeling: The role of an information resource dictionary system, August 1986, Submitted for publication.

[Dolk and Kirsch 1986]

Dolk, D.R. and Kirsch, R.A. A relational information resource dictionary system. Forthcoming in **Communications of the ACM**.

[Geoffrion 1985]

Geoffrion, A.M. **Structured Modeling.** Draft Research Monograph, UCLA Graduate School of Management, January 1985.

[Geoffrion 1986]

Geoffrion, A.M. An introduction to structured modeling. Working Paper No. 338, Graduate School of Management, UCLA, June 1986.

[Kirsch, R. 1985]

Kirsch, R.A. A relational data dictionary compatible with the National Bureau of Standards information resource dictionary system. Naval Postgraduate School Master's Thesis, December 1985.

[Kroenke 1983]

Kroenke, D. **Database Processing**, 2nd edition, Science Research Associates, 1983.

[Leong-Hong and Plagman 1983]

15. Leong-Hong, B.W. and Plagman, B.K. **Data Dictionary/Directory Systems: Administration, Implementation and Usage**. Wiley-Interscience, 1982.

[Martin and Finkelstein 1983]

Martin, J. and Finkelstein, C. **Information Engineering**, Savant Institute, 1983.

[Noel 1985]

Noel, A.F. Prototyping with data dictionaries for requirements analysis. Naval Postgraduate School Master's Thesis, March 1985.

APPENDIX A: FIPS IRDS ENTITY, ATTRIBUTE, AND RELATIONSHIP TYPES

DATA Entity Types

1. **DOCUMENT**: describes instances of human readable data such as tax forms or annual reports.

2. **FILE**: describes collections of records which represent an organization's data such as inventory files.

3. **RECORD**: describes instances of logically associated data such as a payroll record.

4. **ELEMENT**: describes an instance of data such as a social-security-number.

5. **BIT-STRING**: describes a string of binary digits.

6. **CHARACTER-STRING**: describes a string of characters.

7. **FIXED-POINT**: describes exact representations of numeric values.

8. **FLOAT**: describes exact representations of approximate numeric values.

PROCESS Entity Types

9. **SYSTEM**: describes a collection of processes and data such as an accounts-payable-system.

10. **PROGRAM**: describes a particular process such as print-accounts-payable-checks.

11. **MODULE**: describes a group of programs that are logically associated such as a sort-module.

EXTERNAL Entity Types

12. **USER**: describes an individual or organization that is using the IRDS such as the accounting-department

Table A-1: The Core System-Standard Schema Entity Types

Attribute Type	Entity Type USR	SYS	PGM	MDL	FIL	DOC	REC	ELE
ADDED-BY	S	S	S	S	S	S	S	S
(ALLOWABLE-RANGE)								P
ALLOWABLE-VALUE								P
CLASSIFICATION	P	P	P	P	P	P	P	P
CODE-LIST-LOCATION								P
COMMENTS	S	S	S	S	S	S	S	S
DATA-CLASS								S
DATE-ADDED	S	S	S	S	S	S	S	S
DESCRIPTION	S	S	S	S	S	S	S	S
DOCUMENT-CATEGORY						S		
DURATION-VALUE		S	S	S				
DURATION-TYPE		S	S	S				
(IDENTIFICATION-NAME) ALTERNATE-NAME	P	P	P	P	P	P	P	P
LAST-MODIFICATION-DATE	S	S	S	S	S	S	S	S
LAST-MODIFIED-BY	S	S	S	S	S	S	S	S
LOCATION	P	P	P	P	P	P		
NUMBER-OF-LINES-OF-CODE			S	S				
NUMBER-OF-MODIFICATIONS	S	S	S	S	S	S	S	S
NUMBER-OF-RECORDS					S			
RECORD-CATEGORY							S	
SECURITY	S	S	S	S	S	S	S	S
SYSTEM		S						

Table A-2: Core System-Standard Schema Attribute Types
(S=Single attribute; P=multiple attribute)

1. **CONTAINS**: describes a situation where an entity type con- other entity types (ex: Accounts-Payable-File CONTAINS Accounts-Payable-Record).

2. **PROCESSES**: describes a situation where an entity type acts upon another entity type (ex: Payroll-Program PROCESSES Payroll-Record).

3. **RESPONSIBLE-FOR**: describes an association between organizational entity type and other entity types indication organizational responsibility (ex: Accounting-Department RESPONSIBLE-FOR General-Ledger-File).

4. **RUNS**: describes an association between user and process entity types (ex: user RUNS program).

5. **GOES-TO**: describes a situation where one process transfers control to another process (ex: Accounts-Payable-Aging-Program GOES-TO Aging-Report-Program).

6. **DERIVED-FROM**: describes a situation where an entity is derived from another entity (ex: Annual-Report DERIVED-FROM Program-File).

7. **CALLS**: describes a situation where on entity calls another entity (ex: Data-Entry-Program CALLS Aging-Program).

8. **REPRESENTED-AS**: describes associations between ELEMENTs and certain data entities that document the ELEMENTs' format (ex: Employee-Name REPRESENTED-AS Character-String)

Table A-3: Core System-Standard Schema Relationship Types

(Note: The FIPS IRDS expresses relationships as ENTITYTYPE-RELSHIP-ENTITYTYPE, e.g. SYSTEM-CONTAINS-SYSTEM. We choose to represent relationships as **RELSHIP(entitytype,entitytype)** as below.)

CONTAINS(system,system)	PROCESSES(system,file)
CONTAINS(system,program)	PROCESSES(system,document)
CONTAINS(system,module)	PROCESSES(system,record)
CONTAINS(program,program)	PROCESSES(system,element)
CONTAINS(program,module)	PROCESSES(program,file)
CONTAINS(module,module)	PROCESSES(program,document)
CONTAINS(file,file)	PROCESSES(program,record)
CONTAINS(file,document)	PROCESSES(program,element)
CONTAINS(file,record)	PROCESSES(module,file)
CONTAINS(file,element)	PROCESSES(module,document)
CONTAINS(document,document)	PROCESSES(module,record)
CONTAINS(document,record)	PROCESSES(module,element)
CONTAINS(document,element)	PROCESSES(user,file)
CONTAINS(record,record)	PROCESSES(user,document)
CONTAINS(record,element)	PROCESSES(user,record)
CONTAINS(element,element)	PROCESSES(user,element)
RESP_FOR(user,file)	DERIVED_FROM(document,file)
RESP_FOR(user,document)	DERIVED_FROM(document, document)
RESP_FOR(user,record)	
RESP_FOR(user,element)	DERIVED_FROM(document,record)
RESP_FOR(user,system)	DERIVED_FROM(element,file)
RESP_FOR(user,program)	DERIVED_FROM(element,document)
RESP_FOR(user,module)	DERIVED_FROM(element,record)
	DERIVED_FROM(element,element)
RUNS(user,system)	DERIVED_FROM(file,document)
RUNS(user,program)	DERIVED_FROM(file,file)
RUNS(user,module)	DERIVED_FROM(record,document)
	DERIVED_FROM(record,file)
	DERIVED_FROM(record,record)
CALLS(program,program)	GOES_TO(system,system)
CALLS(program,module)	GOES_TO(program,program)
CALLS(module,module)	GOES_TO(module,module)

Table A-4: IRDS Relationships

APPENDIX B: BRIEF SUMMARY OF THE SQL DATABASE LANGUAGE

SQL is a query language for creating, modifying, and retrieving data residing in a relational database. The following is a very brief introduction to SQL. We cover only sufficient details to support the discussion in the report. A fuller treatment of SQL can be found in most database texts (e.g.: [Date 1982] or [Kroenke 1983]).

The following three relations describing a parts-supplier example are used for illustrative purposes:

SUPPLIER(**sid**, sname, status, city)

PARTS(**pid**, pname, color, weight)

SUP_PART(**sid**, **pid**, qty)

Relations are indicated in upper-case and attributes in lower-case. Key attributes are shown in bold-face. Each relation can be thought of as a file with each row of the relation corresponding to a record and each attribute corresponding to a field in the record.

Simple queries are expressed using the SELECT-FROM-WHERE syntax:

```
SELECT attribute(s)
FROM   relation(s)
WHERE  boolean_conditions
```

For example, the SQL equivalent of "list all supplier names from Dallas" is:

```
SELECT SNAME
FROM   SUPPLIER
WHERE  CITY = 'Dallas'
```

Several attributes can be specified in the SELECT clause:

```
SELECT SNAME, SID
FROM   SUPPLIER
WHERE  CITY = 'Dallas'
```

and Boolean conditions can be linked via AND or OR operators:

```
SELECT SNAME, SID
FROM   SUPPLIER
WHERE  CITY = 'Dallas' OR CITY = 'Atlanta'
```

More than one relation can be involved in a query as well. The following query lists all part names supplied by supplier Zukowski:

```
SELECT PNAME
FROM   SUPPLIER, PARTS, SUP_PART
WHERE  SNAME = 'Zukowski' AND SUPPLIER.SID = SUP_PART.SID AND
       SUPPLIER.PID = PARTS.PID
```

When there is uncertainty concerning which relation an attribute in the query belongs to, it's necessary to append the relation prefix to that attribute using the "." separator.

Subqueries can be used to formulate arbitrarily complex queries. The following lists supplier names for suppliers who supply part P2:

```
SELECT SNAME
FROM   SUPPLIER
WHERE  SID IN
      (SELECT SID
       FROM   SUP_PART
       WHERE  PID = 'P2')
```

The subquery (in parentheses) is executed first to identify the set of supplier id's who supply part P2. This requires accessing the SUP_PART relation. Once that set has been identified then the main part of the query is executed to find the corresponding supplier name for each supplier id in the set.

Views can be created which are equivalent to logical files. For example, we may want to create a view of red parts only:

```
CREATE VIEW REDPARTS AS
(SELECT PID, PNAME, WEIGHT
 FROM   PARTS
 WHERE  COLOR = 'red')
```

This view can be manipulated exactly like any other relation. The only difference is that physically this view will not be stored as a separate relation.

APPENDIX C: IRDS MACROS

Most RDBMS allow the user to define macros as a convenient way of invoking specific SQL queries. This frees the user from having to know the underlying syntax of the corresponding command and provides a concise means of executing complex commands. By defining an appropriate set of these macros, the database and model administrator can provide the user with a flexible toolkit for data and model management. This appendix lists a few such macros as they would appear in the ORACLE RDBMS environment. Please note in this context that the "&" has a special meaning in ORACLE unrelated to its use in structured modeling. Specifically, the "&" refers to an argument whose value must be prompted from the user when the macro is invoked.

IMPACT_OF_CHANGE

If we change a specified information resource (&ent_name, &ent_type), what other information resources will this have an impact upon?

```
SELECT ANAME,DNAME FROM ENTITY
WHERE ANAME IN
  (SELECT E1NAME FROM RELSHIP
   WHERE E2NAME='&ent_name' AND E2TYPE='&ent_type')  OR
      ANAME IN
  (SELECT E2NAME FROM RELSHIP
   WHERE E1NAME='&ent_name' AND E1TYPE='&ent_type');
```

REL_INTEGRITY

Display which entity-types can participate in which relationship-types. This is essentially a dump of Table A-4.

```
SELECT RNAME,E1NAME,E2NAME FROM RELSHIP
WHERE E1TYPE='ENT_TYPE' AND E2TYPE='ENT_TYPE';
```

Equivalent relationship-specific macros can be fashioned by substituting the relationship-type for RELSHIP as follows (e.g.: CONTAINS_INTEGRITY):

```
SELECT E1NAME, E2NAME FROM CONTAINS
WHERE E1TYPE='ENT_TYPE' AND E2TYPE='ENT_TYPE';
```

CHECK_INVALID

Identify any invalid relationship instances (i.e., instances which violate relationship-type integrity constraints) in the IRDS:

```
SELECT RNAME,E1NAME,E1TYPE,E2NAME,E2TYPE FROM RELSHIP
WHERE E1TYPE != 'ENT_TYPE' AND E2TYPE != 'ENT_TYPE' AND
      (E1TYPE,E2TYPE) NOT IN
        (SELECT E1NAME,E2NAME FROM RELSHIP
         WHERE E1TYPE = 'ENT_TYPE' AND E2TYPE='ENT_TYPE')
```

Again, this could be made relationship-specific by substituting for RELSHIP. All invalid instances could be deleted by replacing the SELECT ... FROM RELSHIP clause above with DELETE FROM RELSHIP.

CALL_SEQ

Determine proper calling sequences for structured modeling genus types.

```
SELECT E2NAME FROM CALLS
WHERE  E1NAME = '&genus_type' AND E2TYPE = 'ENT_TYPE'
```

GENUS_STRUCTURE

Determine whether the generic structure of a model violates any of the rules of structured modeling ("!" stands for a logical "NOT"):

```
SELECT E1NAME, E1TYPE, E2NAME, E2TYPE FROM CALLS
WHERE E1TYPE != 'ENT_TYPE' AND E2TYPE != 'ENT_TYPE' AND
      MODEL = '&model_name' AND
      (E1TYPE, E2TYPE) NOT IN
      (SELECT E1NAME, E2NAME FROM CALLS
       WHERE E1TYPE = 'ENT_TYPE' AND E2TYPE = 'ENT_TYPE')
```

MODULE_CHECK

Check that no genus belongs to more than one module as follows:

```
SELECT E1NAME, E2NAME FROM CONTAINS
WHERE  E1TYPE = 'MODULE' AND COUNT(*) > 1 AND
       MODEL = '&model_name' AND
       E2TYPE IN ('PE', 'CE', 'ATT', 'VA', 'TEST', 'FCN')
```

ADJACENCY

Determine the adjacency for a specific genus (i.e., its calling sequence) within a model:

```
SELECT E2NAME, E2TYPE
FROM CALLS WHERE E1NAME = '&genus' AND MODEL = '&model_name'
```

NATURAL_LANG

Display a natural language summary of a specific model for management purposes:

```
SELECT ENAME, ETYPE, INDEX, COMMENTS
FROM   GENUS
WHERE  MODEL = '&model_name'
```

MATH_SUMMARY

Display a mathematical summary of a model for modelers:

```
SELECT ENAME, ETYPE, INDEX, INDEX_STMT
FROM   GENUS
WHERE  MODEL = '&model_name'
```

APPENDIX D: FUNDAMENTALS OF STRUCTURED MODELING

Structured modeling is a unified modeling framework based on acyclic, attributed graphs. There are three basic structures which comprise this framework: **elemental**, **generic**, and **modular**.

Models are defined in terms of elements which may be partitioned into genera (pl. of "genus") and further aggregated into modules. There are five element types: **primitive entity**, **compund entity**, **attribute** (plus a variation called a **variable attribute**), **function**, and **test**. Primitive entities are existential in nature and have no value mathematically. Compound entities reference other entities already defined and require no value. Attributes associate a certain property and value with an entity or combination of entities. Variable attributes are like attributes except that values may not be specified. Variable attributes most resemble decision variables in a linear programming model. Function elements associate a rule and value with an entity or combination of entities. Function elements resemble mathematical equations. Test elements are like function elements with a boolean (True,False) value. Test elements constraints in mathematical programmming models.

Each element has a **calling sequence** which identifies other elements directly referenced. The calling sequence captures the cross-references among model elements and can be derived directly from the graphical representation. The **elemental structure** of a model is a nonempty, closed, finite, acyclic collection of elements. Acyclicity implies that there is no sequence of calling sequences which turns out to be "circular".

The **generic structure** of a model is a partitioning of the elemental structure such that there is one partition (genus) for each element type. Genus is similar to the notion of set or class. Partitioning must satisfy **generic similarity** in that every element in a genus must have the same number of calling sequence segments and all elements in a given calling sequence segment must belong to the same genus. Partitioning enforces strong typing in that a single element may belong to one, and only one, genus.

Modular structure is a tree defined on the generic structure all of whose leaves are genera and all of whose non-terminal nodes are modules. Modular structure allows genera to be grouped in ways that might be conceptually meaningful to users. It facilitates a view mechanism which allows users to view the model at different levels of abstraction. Not all modular structures are permitted, however. Only those which satisfy **monotone ordering**, i.e. those which admit an indented list representation with no forward references (genera which call genera further down the list), are allowed.

A **structured model** consists of an elemental stucture, a generic structure which satisfies generic similarity, and a modular structure with monotone ordering.

The easiest way to absorb the terminology is by examining a simple example. The transportation model is a familiar model discussed in all introductory texts on management science. The scenario entails plants which produce a single product for shipment to customers. Every plant has a maximum supply capacity and every customer has an exact demand requirement. For every link which exists between a plant and a customer, there is an associated unit transportation cost. The model allows us to evaluate various transportation flows over the links which satisfy production capacities and demand requirements in terms of the resultant total transportation cost.

Primitive entities include every instance of a plant (assume plants in Dallas and Chicago) and every instance of a customer (assume customers in Seattle, Boston, and Atlanta). Compound entities include every link between a plant and a customer (assume links Dallas-Seattle, Dallas-Atlanta, Dallas-Boston, Chicago-Seattle, and Chicago-Boston). Attributes include the supply capacity for each plant, the demand requirement for each customer, and the transportation cost for each link. The flow for each link is a variable attribute. Test elements consist of the supply constraint for each plant and the demand constraint for each customer. A single function element describes the total transportation cost.

Calling sequences for the transportation model capture the functional dependencies among the model elements. In general, attributes of an entity will have that entity in its calling sequence. Thus each supply capacity has a plant in its calling sequence, each demand requirement a customer, and each transportation flow and cost a link. Compound entities have the entities which they depend upon in their calling sequence thus each link has both a customer and a plant in its calling sequence. The supply and demand constraints depend upon supply and demand respectively as well as the variable attribute transportation flow. Finally, the total cost function element depends upon the cost and flow.

The elemental structure of this model captures all the associations among the elements just described and can be represented as an acyclic graph. In general, this graph will be too detailed to be of much use. We can partition the elemental structure into a generic structure by defining the following genera: primitive entity (PLANT and CUSTOMER), compound entity (LINK), attribute (SUPPLY, DEMAND, COST), variable attribute (FLOW), test (T:SUP and T:DEM), and function (TOTAL_COST). The similarity requirement insures that calling sequences for the genera are exactly as those for the elements. The graphical representation of the generic structure is more concise and meaningful (Figure D-1).

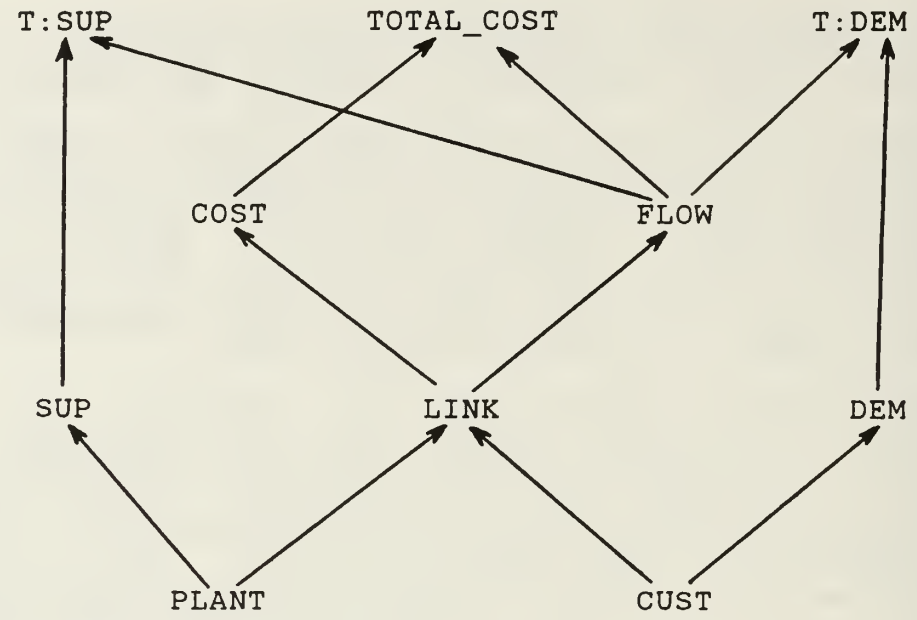


Figure D-1: Generic Structure of Transportation Model
(from [Geoffrion 1985])

A modular structure can be imposed upon the generic structure to represent higher levels of abstraction by grouping related genera. For example, we may want to aggregate segments of the overall model into sales, production, and distribution components. This can be done by defining modules &SALES, &PROD, and &DIST (the "&" by convention refers to a module) which are rooted trees whose leaves are genera (Figure D-2). The modular structure can, in a sense, be viewed as orthogonal to the generic structure while preserving the acyclic nature of the latter. Modular structure facilitates different views of the model corresponding to the level of detail which a user desires. The monotone ordering of the modular structure insures that these different views all preserve the acyclicity of their corresponding generic structures.

The modular structure can be represented as a modular outline where a preorder traversal of the hierarchy is performed and levels of indentation correspond to levels of the hierarchy indentation (Figure D-2). The monotone requirement for modular structures insures that there are no forward references in this outline, i.e. no genus in the outline has any genera in its calling sequence which appear later in the outline. This outline is fleshed out into a model schema by adding relevant information such as genus type, calling sequence, mathematical representation, and natural language interpretation.

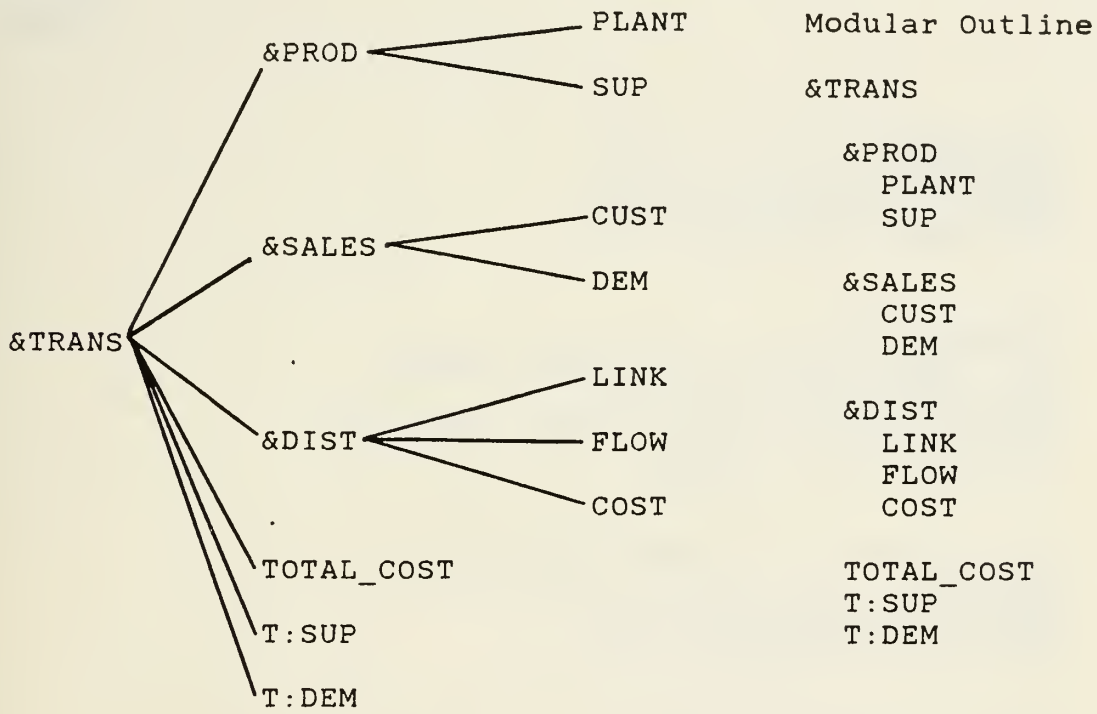


Figure D-2: Modular Structure and Outline for Transportation Model (from [Geoffrion 1985])

DISTRIBUTION LIST

	Number of Copies
Professor Daniel R. Dolk Code 54DK Naval Postgraduate School Monterey, CA 93943	5
Headquarters U.S. Army Military Personnel Center 200 Stovall St. ATTN: DAPC-PLF (Major Robert Trackwell) Alexandria, VA 22332	5
Administrative Sciences Department Code 54 Naval Postgraduate School Monterey, CA 93943	1
Computer Center Library Code 0141 Naval Postgraduate School Monterey, CA 93943	1
Defense Technical Information Center Cameron Station Alexandria, VA 23314	2
Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1